Filter Distribution Templates in Convolutional Networks for Image Classification Tasks

Ramon Izquierdo-Cordova, Walterio Mayol-Cuevas
Department of Computer Science, University of Bristol, United Kingdom
{ramon.izquierdocordova,walterio.mayol-cuevas}@bristol.ac.uk

Abstract

Neural network designers have reached progressive accuracy by increasing models depth, introducing new layer types and discovering new combinations of layers. A common element in many architectures is the distribution of the number of filters in each layer. Neural network models keep a pattern design of increasing filters in deeper layers such as those in LeNet, VGG, ResNet, MobileNet and even in automatic discovered architectures such as NASNet. It remains unknown if this pyramidal distribution of filters is the best for different tasks and constrains. In this work we present a series of modifications in the distribution of filters in three popular neural network models and their effects in accuracy and resource consumption. Results show that by applying this approach, some models improve up to 8.9% in accuracy showing reductions in parameters up to 54%.

1. Introduction

An important consideration to create a convolutional neural network (CNN) model is the number of filters required at every layer. The Neocognitron implementation for example, keeps an equal number of filters for each layer in the model [4]. A very common practice has been to use a bipyramidal architecture. The number of filters across the different layers is usually increased as the size of the feature maps decreases. This pattern was first proposed in [13] with the introduction of LeNet and can be observed in a diverse set of models such as VGG[20], ResNet[6] and MobileNet[8]. Even models obtained from automatic model discovery, like NASNet [24], follow this principle since neural architecture search methods are mainly formulated to search for layers and connections while the number of filters in each layer remains fixed. The motivation behind this progressive increase in the number of kernels is to compensate a possible loss of the representation caused by the spatial resolution reduction [13]. In practice it improves performance by keeping a constant number of operations in each layer [1]. It remains unknown if this pyramidal distribution of filters is also beneficial to different aspects of model performances other than the number of operations.

The contribution of this paper is to challenge the widely used design of increasing filters in neural convolutional models by applying a small subset of diverse filter distributions, called templates, to existing neural network designs. Experimental evidence shows that simple changes to the pyramidal distribution of filters in convolutional network models lead to improvements in accuracy, number of parameters or memory footprint; we highlight that most recent models, which have had a more detailed tuning in the filter distribution, present resiliency in accuracy to changes in the filter distribution, a phenomena that requires further research and explanation.

Experiments in this document are exploratory. We use equal number of filters in all templates without constraining the effects of the redistribution. We extend this work in [9] where templates are evaluated with more rigorous experiments keeping FLOPs to similar values as in the original model and then comparing resource consumption.

2. Related Work

The process of designing a neural network is a task that has largely been based on experience and experimentation which consumes a lot of time and computational resources. Of note are reference models such as VGG[20], ResNet[6], Inception[21] and MobileNet[8] that have been developed with significant use of heuristics. Even with automatic methods, one key feature that has constantly been adopted is the manual selection of the number of filters in each layer in the final model. Filters are set in such a way to have an increasing number as the layers go deeper, differing from the original Neocognitron design[4].

With the increase in the use of Neural Networks, and particularly Convolutional Networks for computer vision problems, a mechanism to automatically find the best architecture has become a requirement in the field of Deep Learning. One of the biggest challenges in automatic architec-

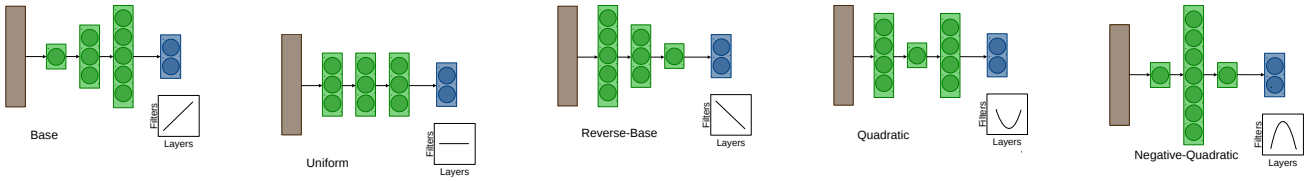

Figure 1. Filters per layer using the proposed templates for filter redistribution in a VGG style model. Base distribution, which is the original distribution, shows the common design of growing the filters when resolution of feature maps decreases in deeper layers. Although the total number of filters is kept constant after templates, changes in filter distribution induce different effects in performance and resource consumption.

ture design is that the search space for CNN architectures is very large[18]. Two fields have derived from the problem: *i)* neural architecture search (NAS), that develops mechanisms for searching for the best combination of layers[25] and *ii)* channel number search (CNS), which look for the best distribution of filters given an initial architecture[2, 22].

Pruning methods could be seen as an special case of CNS in which there is the assumption that the weights, obtained at the end of the training process of the original model, are important to the pruning method[3].

In pruning methods, searching involves training models for several iterations to select the correct weights to remove [3, 7, 23], or at least increasing the computation during the training when doing jointly training and search [12, 15]. In [16] it is suggested that accuracy obtained by pruning techniques can be reached by removing filters to fit a certain resource budget and training from scratch.

Our work for finding an appropriate distribution of filters relates to [5] in the sense that their method is not restricted to reducing filters but also to increase them to see if the changes are beneficial. Our approach differs however, because it only requires the model to be trained in the final stage, after manually making some predefined changes to the number of filters using the redistribution templates.

3. Filter distribution templates

While most of neural network architectures show an incremental distribution of filters, recent pruning methods such as [5, 12], have shown different filter distribution patterns emerging when reducing models like VGG that defy the notion of pyramidal design as the best distribution for a model. This is a motivational insight into what other distributions can and should be considered when designing models. On one side the combinatorial space of distributions make this a challenging exploration, on the other however, it importantly highlights the need to pursue such exploration if gains in accuracy and overall performance can be made.

In this work, rather than attempting to find the optimal filter distribution with expensive automatic pruning or growing techniques, we propose to first adjust the filters of a convolutional network model via a small number of predefined templates. These templates such as those depicted in figure 1, are inspired by existing models that have already been found to perform well and thus candidates that could be beneficial for model performance improvement beyond the number of operations. Performance criteria such as accuracy, memory footprint and inference time are arguably as important as the number of operations required.

In particular, we adopt as one template, a distribution with a fixed number of filters as with the original Neocognitron design, but also other templates inspired by the patterns found in [5] where some behaviours are present in different blocks from the resulting ResNet101 model: 1) filters agglomerate in the centre and 2) filters are reduced in the centre of the block. In [12, 23] is shown also a pattern with more filters in the centre of a VGG model. Based on these observations we define the templates we use in this work.

Different distributions with the same number of filters can lead to different number of parameters (e.g. weights) and different memory or computational requirements (e.g. GPU modules). In the toy example in Figure 2, both models have the same number of filters but the one on the right has less parameters and less compute requirements at the cost of more memory footprint. This example highlights the compromises that filter distributions can offer for the design and operation of network models.

We define a convolutional neural network base model as a set of numbered layers $L = 1, ..., D + 1$, each with f_l filters in layer l, $D + 1$ is the final classification layer. The total number of filters that can be redistributed is given by $F = \sum_{l=1}^{D} f_l$. We want to test if the common heuristic of distributing F having $f_{l+1} = 2f_l$ each time the feature map is halved, is advantageous to the model over other distributions of F when evaluating performance, memory footprint and inference time.

The number of filters in the final layer $D+1$ depends on the task and remains the same for all the templates, therefore it is not taken into account for computing the number of filters to redistribute. For architectures composed of blocks (e.g. Inception) we consider blocks as single layers and keep the number of filters within a block the same. As a result, a final Inception module marked with f_l filters, is set

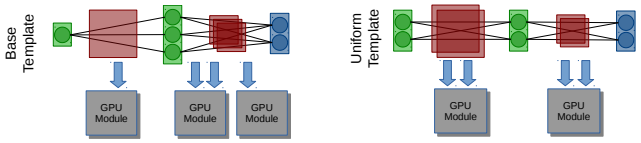

Figure 2. A toy example to show how two different templates with the same number of filters produce a variety of effects in parameters, memory, inference time and flops. Layers (rectangles) contain in total, equal number of filters (circles) for both templates. Lines between filters represent parameters, red squares are by-channel feature maps which reside in memory jointly with parameters. Flops are produced by shifting filters along feature maps. Inference time is affected by flops and number of transfers, indicated by blue arrows and here limited to two simultaneously, between memory and GPU modules. Diagram assumes filters of equal sizes and pooling between layers. Differences are scaled up in real models counting thousand of filters.

to that number of filters in each layer inside the module.

Uniform Template. The most simple distribution to evaluate is, as the original Neocognitron, an uniform distribution of filters. Computing the number of filters in an uniform distribution is straightforward, the new number in each layer is given by $f'_l = F/D \quad \forall l \in \{1, ..., D\}$.

Reverse Template. Another straight-forward transformation for the filter distribution adopted in this paper is reversing the number of filters in every layer. Our final model with this template is defined by the filters $f'_l = f_{D-l+1}$.

Quadratic Template. This distribution is characterised by a quadratic equation $f'_l = al^2 + bl + c$ and consequently, has a parabolic shape with the vertex in the middle layer. We set this layer to the minimal number of filters in the base model $f_{min} = min\,(f_l) \quad l \in \{1, ..., D\}$ so, the number of filters is described by $f'_{D/2} = f_{min}$. Also, we find the maximum value in both the initial and final convolutional layers, thus $f'_1 = f'_D$.

To compute the new number of filters in each layer we solve the system of linear equations given by *i)* the restriction of the total number of filters in $\sum_{l=1}^{D}(f'_l) = \sum_{l=1}^{D}\left(al^2 + bl + c\right) = F$, that can be reduced to $\left(\frac{D^3}{3} + \frac{D^2}{2} + \frac{D}{6}\right)a + \left(\frac{D^2}{2} + \frac{D}{2}\right)b + Dc = F$, *ii)* the equation produced by the value in the vertex $f'_{D/2} = \frac{D}{2}^2 a + \frac{D}{2}b + c = f_{min}$ and *iii)* the equality from the maximum values which reduces to $(D^2 - 1)a + (D - 1)b = 0$.

Negative Quadratic Template. It is a parabola with the vertex in a maximum, that is, a negative quadratic curve. The equation is the same as the previous template but the restrictions change. Instead of defining a value in the vertex, f'_l at the initial and final convolutional layers are set to the minimal number of filters in the base model $f'_l = f_{min} \quad l \in \{1, D\}$. The number of filters in each layer is computed again with a system of equations specified by *i)*, the restriction of the total number of filters as in the quadratic template, and the two points already known in the first and last convolutional layers defined by *ii)* $a + b + c = f_{min}$ and *iii)* $D^2a + Db + c = f_{min}$.

Table 1. Model performances with the original distribution and four templates for the same number of filters evaluated on CIFAR-10, CIFAR-100 and Tiny-Imagenet datasets. After filter redistribution models surpass the base accuracy. Results show average of three repetitions.

Model	Base	Redistribution Templates			
		Rev Base	Unif	Quad	Neg Quad
CIFAR-10					
VGG-19	93.52	**94.40**	94.24	94.18	94.21
ResNet-50	94.70	95.17	95.08	94.41	**95.23**
Inception	94.84	94.60	94.82	**94.86**	94.77
MobileNet	89.52	**91.35**	91.28	89.98	91.04
CIFAR-100					
VGG-19	71.92	**74.65**	74.03	73.55	74.05
ResNet-50	**77.09**	74.80	76.65	75.71	76.76
Inception	78.03	77.78	**78.12**	77.67	76.65
MobileNet	65.08	66.39	**68.71**	63.89	67.05
Tiny-Imagenet					
VGG-19	54.62	57.73	56.68	54.73	**59.50**
ResNet-50	**61.52**	53.67	60.97	59.77	60.12
Inception	54.80	55.24	55.78	54.97	**55.87**
MobileNet	56.29	51.40	**58.11**	53.37	55.76

4. Models Comparison Under Size, Memory Footprint and Speed

In this section we investigate the effects of applying different templates to the distribution of kernels in convolutional neural network models (VGG, ResNet, Inception and MobileNet). We compare models under the basis of size, memory and speed in three of the popular datasets for classification tasks.

Datasets and Models

We trained over three datasets traditionally used for convolutional network evaluation: CIFAR-10, CIFAR-100 [10] and Tiny-Imagenet [11]. The first two datasets contain sets of 50,000 and 10,000 colour images for train and validation respectively, with a resolution of 32x32. Tiny-Imagenet is a reduced version of the original Imagenet dataset with only 200 classes and images with a resolution of 64 x 64 pixels.

We evaluate some of the most popular CNN models: VGG[20], ResNet[6], Inception[21] and MobileNet[8]; which represent some of the highest performing CNNs on the ImageNet challenge in previous years [19].

Implementation Details

Experiments have models fed with images with the common augmentation techniques of padding, random cropping and horizontal flipping. Our experiments were run in a

Table 2. Parameters, memory and inference time for selected models when applying our templates keeping the same number of filters evaluated on the CIFAR-10 (black) and Tiny-Imagenet (blue) datasets. Models are normally optimised to fast GPU operation, therefore the original base distribution has a good effect in speed but the redistribution of filters induced by our templates makes models capabilities improve on the other metrics. Memory footprint reported by CUDA.

Resource	Model	Base		Redistribution Templates							
				Reverse Base		Uniform		Quadratic		Negative Quadratic	
Parameters (Millions)	VGG-19	20.0	25.0	20.0	20.6	16.0	**19.3**	**15.8**	20.7	20.0	20.6
	ResNet-50	23.5	23.9	23.1	23.1	**12.9**	**13.0**	19.0	19.3	33.0	33.0
	Inception	**6.2**	19.2	6.7	**10.0**	**6.2**	12.7	7.2	18.7	7.0	10.1
	MobileNet	3.2	3.4	**2.2**	**2.4**	**2.2**	**2.4**	3.2	3.3	2.4	2.6
Memory Footprint (GB/batch)	VGG-19	**1.3**	1.5	2.6	10.0	4.4	4.8	2.0	6.8	1.4	3.8
	ResNet-50	3.1	5.0	11.5	10.1	4.1	9.6	7.9	7.5	**3.0**	9.8
	Inception	**1.5**	5.8	3.1	10.8	1.7	6.7	2.2	8.6	1.6	5.9
	MobileNet	2.5	2.5	5.1	5.1	1.5	5.9	6.0	4.8	**1.0**	**1.9**
Inference Time (ms/batch)	VGG-19	**3.0**	4.9	8.2	4.1	5.3	4.2	7.5	4.6	7.3	**3.5**
	ResNet-50	46.4	13.3	61.0	12.8	**23.4**	12.8	59.0	**11.0**	47.6	29.9
	Inception	28.5	24.0	54.9	21.4	34.3	28.3	25.2	**18.3**	**24.3**	31.4
	MobileNet	**3.8**	5.8	6.8	6.7	4.3	9.7	7.4	7.3	4.9	**5.3**

NVidia Titan X Pascal 12GB GPU adjusting the batch size to 128. All convolutional models, with and without templates, were trained for 160 epochs using the same conditions. Therefore, there is some margin for improving accuracy for each distribution by performing individual hyperparameter [17, 14]. We used stochastic gradient descent (SGD) with weight decay of 1e-4, momentum of 0.9 and a scheduled learning rate starting in 0.1 for the first 80 epochs, 0.01 for the next 40 epochs and finally 0.001 for the remaining epochs.

Template effect over baseline models

We conducted an experiment to test our proposed templates on the selected architectures. Table 1 shows VGG, Inception and MobileNet accuracies improving in all datasets when templates are applied. Being complex architectures, ResNet and Inception present the highest accuracy in general. A surprising finding is that in both models difference in accuracy between templates is less than 2.3% despite the drastic modifications that models are suffering after the change of filter distribution. Models that share a sequential classical architecture such as VGG and MobileNet, show a better improvement when using templates in Tiny-Imagenet. A remarkable accuracy improvement of 4.88 percentage point is achieved in VGG.

When analysing resource consumption (Table 2), we find models are affected differently with each template and model. Reverse-Base, Uniform and Quadratic templates show some reductions in the number of parameters while Negative Quadratic template reduces the memory usage. Inference Time is affected negatively for most of the templates. This is an expected result as original models are designed to perform well in the GPU. Inception model shows an improvement in speed with reductions of 14% over inference time respect to the base model while maintaining comparable accuracy. ResNet is able to reduce inference time in 49% at the cost of having slightly less accuracy than the base model.

5. Conclusions

The most common design of convolutional neural networks when choosing the distribution of the number of filters is to start with a few and then to increase the number in deeper layers. We challenged this design by evaluating some architectures with a varied set of distributions on the CIFAR and Tiny-Imagenet datasets. Our results suggest that this pyramidal distribution is not necessarily the best option for obtaining the highest accuracy or even the highest parameter efficiency.

Our experiments show that models, with the same amount of filters but different distributions produced by our templates, improve accuracy with up to 4.8 points for some model-task pairs. In terms of resource consumption, they can obtain a competitive accuracy compared to the original models using less resources with up to 56% less parameters and a memory footprint up to 60% smaller. Results also reveal an interesting behaviour in evaluated models: a strong resilience to changes in filter distribution. The variation in accuracy for all models after administering templates is less than 5% despite the considerable modifications in the distributions and therefore, in the original design. Our work overall offers insights to model designers, both automated and manual, to construct more efficient models by introducing the idea of new distributions of filters for neural network models and help gather data to build understanding of the design process for model-task pairs.

Acknowledgments

This work was partially supported by CONACYT and the Secretaría de Educación Pública, México.

References

[1] Joseph Lin Chu and Adam Krzyżak. Analysis of feature maps selection in supervised learning using convolutional neural networks. In *Canadian Conference on Artificial Intelligence*, pages 59–70. Springer, 2014. 1

[2] Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. *arXiv preprint arXiv:1905.09717*, 2019. 2

[3] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018. 2

[4] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980. 1

[5] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1586–1595, 2018. 2

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 3

[7] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2019. 2

[8] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 1, 3

[9] Ramon Izquierdo-Cordova and Walterio Mayol-Cuevas. Towards efficient convolutional network models with filter distribution templates. https://research-information.bris.ac.uk/en/persons/walterio-w-mayol-cuevas, 2021. 1

[10] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 3

[11] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7:7, 2015. 3

[12] Guillaume Leclerc, Manasi Vartak, Raul Castro Fernandez, Tim Kraska, and Samuel Madden. Smallify: Learning network size while training. *arXiv preprint arXiv:1806.03723*, 2018. 2

[13] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1

[14] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017. 4

[15] Yawei Li, Shuhang Gu, Luc Van Gool, and Radu Timofte. Learning filter basis for convolutional neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5623–5632, 2019. 2

[16] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018. 2

[17] Gaurav Mittal, Chang Liu, Nikolaos Karianakis, Victor Fragoso, Mei Chen, and Yun Fu. Hyperstar: Task-aware hyperparameters for deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8736–8745, 2020. 4

[18] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *arXiv preprint arXiv:2006.02903*, 2020. 2

[19] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 3

[20] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 3

[21] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 1, 3

[22] Jiaxing Wang, Haoli Bai, Jiaxiang Wu, Xupeng Shi, Junzhou Huang, Irwin King, Michael Lyu, and Jian Cheng. Revisiting parameter sharing for automatic

neural channel number search. *Advances in Neural Information Processing Systems*, 33, 2020. 2

[23] Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 2130–2141, 2019. 2

[24] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017. 1

[25] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018. 2